In the Office Action, the Examiner rejected claims 1-7 and 10-41 under 35 USC §103. These objections and rejections are fully traversed below. Claims 1-7 and 10-41 remain pending.

Reconsideration of the application is respectfully requested based on the following remarks.

REJECTION OF CLAIMS UNDER 35 USC §103

In the Office Action, the Examiner rejected claims 1-4, 6-7, and 18-26 under 35 USC §103 as being unpatentable over Anderson et al, U.S. Patent No. 5,448,735, ('Anderson' hereinafter) in view of Munro (Writing DLLs for Windows using Visual Basic, part 1). This rejection is fully traversed below.

Claim 1 relates to DLLs. Anderson neither discloses nor suggests applying the claimed method to DLLs. In fact, in the Background section of Applicant's specification, various prior art methods of making library calls are addressed. As one example, "calling" and "returning" in a manner similar to subroutine calls is viewed as inefficient. Anderson fails to address the problems associated with calling DLLs or a motivation to reduce the required execution time of DLLs. In addition, since DLLs are not loaded until they are needed, DLLs pose specific issues that do not exist with the system of Anderson.

Claim 1 recites, in relevant part, "wherein each of the code modules responsible for calling a next one of the code modules in the chain includes a reference to the next one of the code modules in the chain, wherein an address in memory at which the next one of the code modules in the chain is loaded is associated with the reference to the next one of the code modules in the chain". Anderson neither discloses nor suggests the invention of claim 1. Specifically, Anderson neither discloses nor suggests "wherein each of the code modules responsible for calling a next one of the code modules in the chain includes a reference to the next one of the code modules in the chain, wherein an address in memory at which the next one of the code modules in the chain is loaded is associated with the reference to the next one

of the code modules in the chain." (Emphasis added). Anderson merely discloses the use of pointers to reference code modules. Anderson neither discloses nor suggests associating an address in memory at which the next one of the code modules in the chain is loaded with the reference to the next one of the code modules in the chain.

The Examiner admits that Anderson does not teach that the one or more code modules are one or more DLLs. The Examiner seeks to cure the deficiencies of Anderson with Munro.

Munro is a tutorial for writing DLLs for Windows using Visual Basic. See Title. In this tutorial, Munro discloses that "DLLs are special libraries that load into memory only once, at run-time, and can thereafter be used by many programs." See page 1, lines 31-33. The tutorial indicates that a main module is implemented. See page 3, lines 28-42. The tutorial further indicates that, "[w]hile it has its own data segment, a DLL does not have a stack. Unlike DOS applications, in which the stack segment and the data segment are the same, a DLL shares the stack segment with the calling program." See page 4, lines 25-28. A DLL "has its own data segment, which is allocated by Windows when the DLL is loaded..." See page 6, lines 28-29. A DLL procedure is responsible for preserving the old DS and moving AX into DS for use by the program. See page 6, lines 30-37. "A DLL is similar in structure to an executable program under Windows. It is created by using the linker to pull together all the various modules. During the process of linking, a definition file is used to specify which files are exports..." See page 7, lines 27-30. The DLL's properties include a STUB. See page 7, lines 41-43.

As set forth on page 2, lines 3-15 of the background section of Applicant's specification, "A DLL is an executable file that traditionally cannot be run independently. In other words, a DLL can only be run by being called from another executable file. This is typically accomplished through subroutine calls. For instance, the executable for the main program may be compiled with a library of "stubs" which allow link errors to be detected at compile-time." The disadvantages of performing subroutine calls are described on page 2, lines 9-15, which include the amount of overhead involved in calling each subroutine and returning to the main program.

The claimed invention enables DLLs to be called without being called by a main program. As a result, the time that is typically required to call each code module and return to the main program is eliminated.

Munro describes a conventional DLL system as set forth in the background section of Applicant's specification. Specifically, Munro indicates that a main module is implemented, and the use of a linker to pull together the various modules. Moreover, the DLLs are described as including a STUB. When a DLL is called, the DLL preserves the old DS, as is typically performed via a subroutine called by a main program. As such, Munro teaches away from creating a "chain" of code modules (DLLs) for execution, as recited in the pending claims.

It is also important to note that Munro neither discloses nor suggests the disadvantages of calling DLLS from a main program. Similarly, Munro neither discloses nor suggests a solution such as that claimed. As such, there fails to be a motivation to combine the cited references. Accordingly, Applicant respectfully submits that claim 1 is patentable over the cited references.

The above arguments are equally applicable to independent claims 19, 22 and 25. Dependent claims 2-7, 18, 20, and 23-24 depend from one of the independent claims and are therefore patentable for at least the same reasons. However, the dependent claims recite additional limitations that further distinguish them from the cited references. Hence, it is submitted that the dependent claims are also patentable over the cited references.

For instance, with respect to claims 6, 7, and 21, the Examiner admits that Anderson does not explicitly teach determining one or more code modules to be executed to complete configuration of a hardware interface of a router. Anderson fails to disclose configuring a hardware interface of a router. In fact, while Anderson discloses tasks that are to be executed by a processor, Anderson neither discloses nor suggests a method for configuring a device such as a router. Applicant respectfully submits that it would not have been obvious to apply the task organization features of Anderson to configure a hardware interface of a router. Accordingly, Applicant respectfully submits that claims 6, 7 and 21 are allowable over the cited art.

Similarly, with respect to claims 18 and 20, the Examiner admits that Anderson does not explicitly teach associating one of the one or more code modules with a hardware interface to identify a starting point for execution upon occurrence of an interrupt. Col. 9, lines 52-68 discloses DSP devices capable of being used for certain tasks. However,

Anderson fails to disclose or suggest using such tasks to configure an interface of one of these devices or to execute the modules upon occurrence of an interrupt. Accordingly, Applicant respectfully submits that claims 18 and 20 are allowable over the cited art.

In the Office Action, the Examiner rejected claim 5 under 35 USC §103 as being unpatentable over Anderson in view of Munro and further in view of Crick et al, U.S. Patent No. 5,781,797 ('Crick' hereinafter). This rejection is full traversed.

Crick fails to cure the deficiencies of the primary references. Accordingly, Applicant respectfully submits that claim 5 is patentable over the cited references.

In the Office Action, the Examiner rejected claims 10-14, 17, and 27-41 under 35 USC §103 as being unpatentable over Anderson in view of Munro and further in view of Mattson Jr., U.S. Patent No. 6,317,870, ('Mattson' hereinafter). This rejection is fully traversed below.

With respect to claims 10-12, as discussed above, Anderson fails to disclose the presently claimed invention. In addition, the Examiner admits that Anderson does not explicitly teach when it is determined that the first one of the one or more code modules is to subsequently execute a second one of the one or more code modules, updating a branch table of the first one of the one or more code modules to identify an address at which the second one of the one or more code modules is loaded such that the reference to the second one of the one or more code modules in the branch table of the first one of the one or more code modules is associated with the address at which the second one of the one or more code modules is loaded. The Examiner seeks to cure the deficiencies of Anderson with Mattson.

Mattson fails to cure the deficiencies of the primary references. In fact, Mattson relates to "optimization of inter-module procedure calls." See title. Thus, one procedure calls another procedure, and presumably must return to the procedure that called it upon completion of its execution. In other words, Mattson neither discloses nor suggests a chain as claimed (or disclosed in Applicant's specification and illustrated in FIG. 1 of Applicant's specification). Specifically, col. 5, lines 43-65 of Mattson disclose an intermodule call that

calls an import stub. The call instruction is modified to include a call with an offset of the program counter plus the displacement to the import stub, as well as the displacement from the import stub to the function being called. In other words, an address at which the function has been loaded is not directly identified. By definition, a table may include one or more entry. However, the call of Mattson is a single call instruction rather than a "branch table" (which is capable of including more than one branch or jump instruction). Moreover, Mattson merely discloses an optimized manner of implementing a procedure call. As set forth in Applicant's background section, a DLL is typically called via subroutine calls through the use of "stubs." The disadvantages of such an implementation (e.g., overhead in calling a procedure) are set forth on p. 2, lines 3-15 of Applicant's specification. Since an inter-module procedure call would require returning to the calling procedure, similar to a main program subroutine call as described in the Background section of Applicant's specification, the combination of the cited references would fail to achieve the desired result. Moreover, Mattson disclose patching call sites during execution rather than at load time. See col. 2, lines 48-60. As a result, call sites which are never executed will not be patched. In contrast, the present invention updates the branch table, preferably at load time, for each "child" DLL capable of being called even if it may not necessarily be executed as recited in claims 11 and 12. As such, Mattson teaches away from the claimed invention. Crick fails to cure the deficiencies of Anderson and Mattson. Accordingly, Applicant respectfully submits that claims 10-12 are patentable over the cited art.

With respect to claim 13, the Examiner admits that Anderson does not explicitly teach wherein the branch table of the first one of the one or more code modules includes the reference to the second one of the one or more code modules prior to loading the code modules and includes the address of the second one of the one or more code modules after the code modules have been loaded. As set forth above, Mattson fails to cure the deficiencies of the primary reference. Neither of the cited references, separately or in combination, discloses or suggests modifying a branch table entry to include an address at which a DLL has been loaded. Moreover, neither of the cited references discloses or suggests the necessity of including a reference to a DLL in a branch table entry and updating the entry after the DLL has been loaded to include the load address. Accordingly, Applicant respectfully submits that claim 13 is allowable over the cited art.

With respect to claim 14, the Examiner admits that Anderson does not explicitly teach updating a branch table by modifying an entry in the branch table such that a dummy address is replaced with the address of the second one of the one or more code modules. Mattson fails to cure the deficiencies of the primary reference. Neither of the cited references, separately or in combination, discloses or suggests replacing a dummy address with an address at which a DLL has been loaded. Accordingly, Applicant respectfully submits that claim 14 is allowable over the cited art.

The dependent claims recite additional limitations that further distinguish them from the cited references. For instance, as recited in claims 11-17, a branch table of a calling DLL is updated to include an address at which the called DLL is loaded.

In the Office Action, the Examiner rejected claims 15-16 under 35 USC §103 as being unpatentable over Anderson in view of Munro and Mattson Jr., U.S. Patent No. 6,317,870, ('Mattson' hereinafter), and further in view of Crick et al. This rejection is fully traversed below.

With respect to claim 15, two or more executable chains may be identified by a separate branch table entry and a separate address. The Examiner admits that Anderson fails to teach when the first one of the one or more code modules is shared by two or more executable chains of code modules, associating the second one of the one or more code modules with one of the two or more executable chains such that the branch table of the first one of the one or more code modules includes at least two entries, each of the entries identifying one of the two or more executable chains, each of the entries including an address. As set forth above, Munro teaches away from the claimed invention. Mattson and Crick fail to cure the deficiencies of the primary references.

Claim 16 depends from claim 15, and further recites that a parameter is associated with one of two or more executable chains such that each of the entries further includes a parameter used to select one of the two or more executable chains. The Examiner admits that Anderson fails to teach the use of a parameter to select one of the two or more executable chains. Neither of the cited references, separately or in combination, discloses or suggests including a branch table in a calling DLL that is updated to include an address at which the

called DLL is loaded. Moreover, neither of the cited references, separately or in combination, discloses or suggests accommodating different potential executable paths (i.e., chains) in different branch table entries as recited in claim 15, or distinguishing the executable chains by a parameter that is used to select one of the executable chains as recited in claim 16. Accordingly, Applicant respectfully submits that claims 15 and 16 are allowable over the cited art.

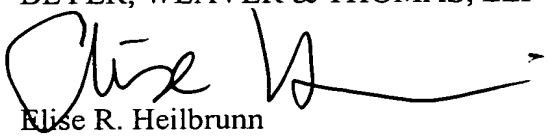
Based on the foregoing, it is submitted that the claims are patentable over the cited references. The additional limitations recited in the independent claims or the dependent claims are not further discussed as the above discussed limitations are clearly sufficient to distinguish the claimed invention from the cited references. Thus, it is respectfully requested that the Examiner withdraw the rejection of claims under 35 USC §103.

SUMMARY

If there are any issues remaining which the Examiner believes could be resolved through either a Supplemental Response or an Examiner's Amendment, the Examiner is respectfully requested to contact the undersigned attorney at the telephone number listed below.

Applicants hereby petition for an extension of time which may be required to maintain the pendency of this case, and any required fee for such extension or any further fee required in connection with the filing of this Amendment is to be charged to Deposit Account No. 50-0388 (Order No. CISCP125).

Respectfully submitted,
BEYER, WEAVER & THOMAS, LLP



Elise R. Heilbrunn
Reg. No. 42,649

BEYER, WEAVER & THOMAS, LLP
P.O. Box 778
Berkeley, California 94704
Tel. (510) 843-6200